Prednáška 5 - Jednoduché pravidlá, Chomského normálny tvar, CYK algoritmus

Ing. Viliam Hromada, PhD.

C-510
Ústav informatiky a matematiky
FEI STU

viliam.hromada@stuba.sk

Transformácie bezkontextových gramatík

- Minulý týždeň sme sa začali baviť o transformáciách gramatík
- Spomenuli sme úpravu odstránenia nadbytočných symbolov gramatiky, t.j. jej úpravu do tzv. redukovaného tvaru.
- Rovnako sme spomínali úpravu odstránenia ε-pravidiel
- Dnes si uvedieme ďalšiu úpravu - odstránenie jednoduchých pravidiel.
- Ukážeme si úpravu gramatiky do Chomského normálnej formy.
- A jej praktické využitie v CYK algoritme.

Motivácia úprav gramatiky

- Nech je daná bezkontextová gramatika $G = (N, T, P, S)$ a reťazec $w \in T^*$ zložený z terminálnych symbolov.
- Uvažujme rozhodovací problém: $w \in L(G)$?. Teda, má reťazec w v gramatike G deriváciu?
- Existujú algoritmy, ktoré dokážu riešiť tento problém s **polynomiálnou zložitosťou** (pre bezkontextové a regulárne gramatiky).
- Pre ich aplikáciu je však často potrebné upraviť príslušnú gramatiku G na ekvivalentnú gramatiku spĺňajúcu isté ďalšie vlastnosti - a to vieme docieliť pomocou spomínaných úprav (ako napr. odstránenie ε-pravidiel alebo odstránenie **jednoduchých pravidiel**).

Odstránenie jednoduchých pravidiel a cyklov

Definícia

Nech $G = (N, T, P, S)$ je bezkontextová gramatika. Potom pravidlá tvaru $A \to B$, kde $A, B \in N$ sa nazývajú ***jednoduché pravidlá****.*

Odstránenie jednoduchých pravidiel

- Každé jednoduché pravidlo typu $A \to B$ odstránime a zároveň do gramatiky doplníme ďalšie pravidlá, aby sa nezmenil výsledný jazyk generovaný gramatikou.
- Zjednodušene, ak sa z A dalo odvodiť B, tak po odstránení pravidla $A \to B$ musíme doplniť pravidlá, ktoré majú na ľavej strane A a na pravej strane všetky možné pravé strany pravidiel typu $B \to$

Príklad

Gramatiku s jednoduchými pravidlami

$S' \to S \mid \varepsilon$

$S \to aSb \mid A \mid ab$

$A \to bAa \mid ba$

Upravíme na:

$S' \to aSb \mid bAa \mid ba \mid ab \mid \varepsilon$

$S \to aSb \mid bAa \mid ba \mid ab$

$A \to bAa \mid ba$

Odstránenie jednoduchých pravidiel

Pre každý neterminál A vytvoríme množinu N_A neterminálov, ktoré sa z A dajú odvodiť, t.j.:

$$N_A = \{B \mid A \Rightarrow^* B, B \in N\} \tag{1}$$

Vytvorenie množiny N_A

Vstup: bezkontextová gramatika $G = (N, T, P, S)$ bez ε-pravidiel, neterminál A
Výstup: množina N_A (4)

1: $N_A \leftarrow \{A\}$
2: **opakuj**
3: $\quad \acute{N_A} \leftarrow N_A$;
4: $\quad N_A \leftarrow \acute{N_A} \cup \{C \mid B \rightarrow C \in P \wedge B \in \acute{N_A}\}$
5: **pokiaľ** $N_A \neq \acute{N_A}$

Odstránenie jednoduchých pravidiel

- Na základe množín N_A teraz môžeme odstrániť jednoduché pravidlá.
- Pre každý neterminál vieme, na aké rôzne iné neterminály sa dá prepísať.
- Ak chceme odstrániť jednoduché pravidlá, musíme znovu zabezpečiť, aby to nezmenilo výsledný jazyk, ktorý gramatika generuje.
- To docielime tak, že pre všetky $B \in N_A$ a všetky pravidlá $B \to \alpha$, ktoré nie sú jednoduché, pridáme do upravenej množiny pravidiel aj pravidlá $A \to \alpha$.

Odstránenie jednoduchých pravidiel - príklad

Nech je daná gramatika (bola výsledkom úpravy odstránenia ε-pravidiel):

$\acute{S} \to S \mid \varepsilon$

$S \to aAbB \mid AC \mid abB \mid A \mid C$

$A \to Aa \mid a$

$B \to bBb \mid ab$

$C \to aCA \mid A \mid aC \mid aA \mid a$

Pre jednotlivé neterminály sú množiny N_A (postupne):

$\acute{S} : \{\acute{S}\}, \{\acute{S}, S\}, \{\acute{S}, S, A, C\} = N_{\acute{S}}$

$S : \{S\}, \{S, A, C\} = N_S$

$A : \{A\} = N_A$

$B : \{B\} = N_B$

$C : \{C\} = \{C, A\} = N_C$

Odstránenie jednoduchých pravidiel - príklad

Po odstránení jednoduchých pravidiel dostávame pravidlá:

$\acute{S} \to \varepsilon \mid aAbB \mid AC \mid abB \mid Aa \mid a \mid aCA \mid aC \mid aA$

$S \to aAbB \mid AC \mid abB \mid Aa \mid a \mid aCA \mid aC \mid aA$

$A \to Aa \mid a$

$B \to bBb \mid ab$

$C \to aCA \mid aC \mid aA \mid a \mid Aa$

T.j. gramatiku bez jednoduchých pravidiel.

Praktické použitie úprav

Spomenuli sme teda 3 typy úprav bezkontextových gramatík:

- Odstránenie ε-pravidiel
- Odstránenie jednoduchých pravidiel
- Odstránenie nadbytočných a nedostupných symbolov.

Tieto 3 typy úprav v praxi používame napríklad pri tvorbe **Chomského normálneho tvaru gramatiky**.

Chomského normálny tvar (forma) gramatiky

Definícia

*Bezkontextová gramatika $G = (N, T, P, S)$ je v **Chomského normálnom tvare**, ak každé pravidlo je v jednom z tvarov:*

1. $A \to BC$ *pre* $A \in N; B, C \in (N - \{S\})$;
2. $A \to a$ *pre* $A \in N, a \in T$;
3. $S \to \varepsilon$.

Úprava do CHNT

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$
Výstup: Gramatika G v CHNT

1: Odstráň z gramatiky ε-pravidlá.
2: Uprav gramatiku, aby počiatočný neterminál nebol v pravej strane žiadneho pravidla.
3: Odstráň z gramatiky jednoduché pravidlá.
4: Odstráň z gramatiky nadbytočné a nedostupné symboly.
5: **pre všetky** pravidlá tvaru $A \to X_1 X_2 ... X_n$, $n \geq 3$ **rob**
6: nahraď pravidlami $A \to X_1 A_1, A_1 \to X_2 A_2, ..., A_{n-2} \to X_{n-1} X_n$, kde $A_1, ..., A_{n-2}$ sú nové neterminály.
7: **koniec pre**
8: **pre všetky** pravidlá tvaru $A \to Y_1 Y_2$, kde $Y_1 \in T \vee Y_2 \in T$ **rob**
9: nahraď na pravej strane pravidiel terminály $Y_1 (Y_2)$ novými neterminálmi $V_1 (V_2)$ a pridaj pravidlo (pravidlá) $V_1 \to Y_1$, resp. $V_2 \to Y_2$
10: **koniec pre**

Krok č. 2 z algoritmu

V algoritme sa po odstránení ε-pravidiel robí úprava: *Uprav gramatiku, aby počiatočný neterminál nebol v pravej strane žiadneho pravidla.*.

- V prípade, že sa pri odstránení ε pridal do gramatiky nový neterminál $\acute{S}$ a pravidlá $\acute{S} \to \varepsilon \mid S$, tak tento krok je irelevantný.
- V prípade, že sa v gramatike jej **aktuálny** počiatočný neterminál S nachádza v niektorej pravej strane niektorého pravidla, **pridá sa nový počiatočný neterminál** $\acute{S}$ a pravidlo $\acute{S} \to S$.

CHNT - príklad

Gramatika $G = (\{S, A\}, \{a, b, c\}, P, S)$ s pravidlami:

$S \to aASa \mid c$

$A \to Abc \mid c$

1) ε-pravidlá nie je potrebné odstrániť - žiadne neobsahuje.
2) Zavedieme symbol $\acute{S}$ a pridáme pravidlo:

$\acute{S} \to S$

$S \to aASa \mid c$

$A \to Abc \mid c$

CHNT - príklad (pokr.)

3,4) Po odstránení jednoduchých pravidiel a nadbytočných symbolov:

$\acute{S} \to aASa \mid c$

$S \to aASa \mid c$

$A \to Abc \mid c$

5-7) Úprava pravidiel dlhších ako 3 symboly:

$\acute{S} \to aA_1 \mid c$

$S \to aA_1 \mid c$

$A \to AA_3 \mid c$

$A_1 \to AA_2$

$A_2 \to Sa$

$A_3 \to bc$

CHNT - príklad (pokr.)

8-10) Odstránenie terminálov z pravidiel s pravou stranou dĺžky 2

$\acute{S} \to V_1 A_1 \mid c$

$S \to V_1 A_1 \mid c$

$A \to A A_3 \mid c$

$A_1 \to A A_2$

$A_2 \to S V_1$

$A_3 \to V_2 V_3$

$V_1 \to a$

$V_2 \to b$

$V_3 \to c$

CHNT - poznámky

- Ak vieme v danej gramatike odvodiť slovo w, potom dĺžka jeho odvodenia v CHNT je $2|w| - 1$.
- Strom odvodenia (derivačný strom) slova v gramatike v CHNT je binárny strom.

Praktické využitie Chomského normálneho tvaru

- Chomského normálny tvar sa v praxi využíva v tzv. CYK algoritme.
- CYK algoritmus je **polynomiálnym** algoritmom na zistenie, či **ľubovoľná bezkontextová gramatika** $G = (N, T, P, S)$ generuje zadaný **reťazec terminálov** $w \in T^*$, teda či $w \in L(G)$.

CYK algoritmus

- Predpokladajme, že máme danú bezkontextovú gramatiku $G = (N, T, P, S)$ v **Chomského normálnom tvare**.
- Chceme zistiť, či dané slovo $w = a_1 ... a_n \in L(G), w \in T^*$.
- Algoritmus CYK (Cocke-Younger-Kasami) to vie zistiť so zložitosťou $O(n^3 \cdot |P|)$, kde n je dĺžka analyzovaného slova pre **hocijakú** bezkontextovú gramatiku a $|P|$ počet pravidiel gramatiky.
- Základnou ideou algoritmu je konštrukcia množín:

$$N_{i,j} = \{A \mid A \Rightarrow^* a_i a_{i+1} ... a_j, A \in N\},$$

  $1 \leq i \leq j \leq n$.
- Ak $S \in N_{1,n}$ potom slovo $w \in L(G)$.

CYK algoritmus

- Postupne sa hľadajú neterminály, ktoré vedia generovať podreťazce daného slova w.
- Na začiatku sa určia neterminály generujúce podreťazce dĺžky 1.
  - Keďže gramatika je v CNF, dané neterminály sa určia priamo z pravidiel.
- Následne sa využíva nasledovná vlastnosť:
  - $B \in N_{j,k}$, t.j. $B \Rightarrow^* a_j ... a_k$
  - $C \in N_{k+1,j+i}$, t.j. $C \Rightarrow^* a_{k+1} ... a_{j+i}$,
    a existuje pravidlo $A \rightarrow BC$, potom $A \in N_{j,j+i}$, pretože
    $A \Rightarrow BC \Rightarrow^* a_j ... a_k a_{k+1} ... a_{j+i}$.

CYK algoritmus I

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$ v CNF, slovo $w \neq \varepsilon$
Výstup: Zistenie, či $w \in L(G)$

1: $w = a_1 ... a_n$
2: **pre všetky** $i \in \{1, ..., n\}$ **rob**
3: **pre všetky** $j \in \{i, ..., n\}$ **rob**
4: $N_{i,j} \leftarrow \emptyset$
5: **koniec pre**
6: **koniec pre**
7: **pre všetky** $i \in \{1, ..., n\}$ **rob**
8: **pre všetky** pravidlá tvaru $A \rightarrow a_i$ **rob**
9: $N_{i,i} \leftarrow N_{i,i} \cup \{A\}$
10: **koniec pre**

CYK algoritmus II

11: **koniec pre**

12: **pre všetky** $i \in \{1, 2, ..., n-1\}$ **rob**
13: **pre všetky** $j \in \{1, 2, ..., n-i\}$ **rob**
14: **pre všetky** $k \in \{j, ..., j+i-1\}$ **rob**
15: **pre všetky** pravidlá $A \to BC$, kde $B \in N_{j,k}$ a zároveň $C \in N_{k+1,j+i}$ **rob**
16: $N_{j,j+i} \leftarrow N_{j,j+i} \cup \{A\}$
17: **koniec pre**
18: **koniec pre**
19: **koniec pre**
20: **koniec pre**
21: **ak** $S \in N_{1,n}$ **potom**
22: **vráť** „Slovo patrí do jazyka. “

CYK algoritmus III

23: **inak**
24: **vráť** „Slovo nepatrí do jazyka. “
25: **koniec ak**

CYK algoritmus - príklad

Je daná gramatika:

$S \to AB \mid AC \mid a$

$A \to DA \mid a$

$B \to b \mid d$

$C \to BC \mid c$

$D \to d$

Zistite pomocou CYK algoritmu, či slovo $dab \in L(G)$.

CYK algoritmus - príklad (pokr.)

Najprv sa zostroja množiny

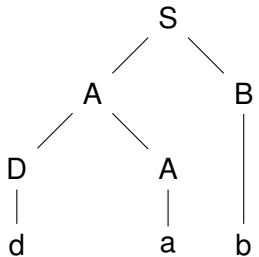
$N_{1,1} = \{B, D\}$, kvôli pravidlám $B \to d$ a $D \to d$.
$N_{2,2} = \{S, A\}$, kvôli pravidlám $S \to a$ a $A \to a$.
$N_{3,3} = \{B\}$, kvôli pravidlu $B \to b$.

Ďalej podľa iterácii:

$i = 1, j = 1, k = 1$	$A \to DA$, $D \in N_{1,1}$, $A \in N_{2,2}$ teda $A \in N_{1,2}$
$i = 1, j = 2, k = 2$	$S \to AB$, $A \in N_{2,2}$, $B \in N_{3,3}$ teda $S \in N_{2,3}$
$i = 2, j = 1, k = 1$	$N_{1,1} = \{B, D\}, N_{2,3} = \{S\}$ do $N_{1,3}$ nepribudne nič
$i = 2, j = 1, k = 2$	$S \to AB$, $A \in N_{1,2}$, $B \in N_{3,3}$ teda $S \in N_{1,3}$

Keďže $S \in N_{1,3}$, tak $dab \in L(G)$.

Na základe príslušných množín vieme teoreticky aj nájsť derivačný strom, resp. odvodenie:



- Základnou úlohou CYK je zistiť, či slovo patrí do jazyka generovaného gramatikou, avšak v podstate sa dá použiť aj na zostrojenie derivačného stromu (na základe množín $N_{i,j}$).
- Výhodou CYK algoritmu je, že vie spracovať všetky bezkontextové gramatiky.
  - Aj nejednoznačné, aj jednoznačné
  - Aj nedeterministické, aj deterministické
- Nevýhodou CYK algoritmu je, že je potrebné, aby gramatika bola v CHNT a taktiež má kubickú zložitosť vzhľadom na dĺžku vstupného slova.
- Existujú aj ďalšie algoritmy, ktoré síce nevedia spracovať všetky bezkontextové gramatiky, avšak majú lineárnu zložitosť vzhľadom na dĺžku vstupného slova.

Použitá literatúra

Aho, A., Lam, M., Sethi, R., Ullman, J.: *Compilers: Principles, techniques and tools.*

Dedera, Ľ: *Počítačové jazyky a ich spracovanie*.

Linz, P.: *An Introduction to Formal Languages and Automata*.